Hindawi
Security and Communication Networks
Volume 2017, Article ID 2713595, 13 pages
https://doi.org/10.1155/2017/2713595

Research Article

CP-ABE Access Control Scheme for Sensitive Data Set Constraint with Hidden Access Policy and Constraint Policy

Nurmamat Helil[1,2] and Kaysar Rahman[1]

[1]*College of Mathematics and System Science, Xinjiang University, Xinjiang, China*
[2]*Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, USA*

Correspondence should be addressed to Nurmamat Helil; nur924@sina.com

Received 15 June 2017; Accepted 23 August 2017; Published 28 September 2017

Academic Editor: Huaizhi Li

CP-ABE (Ciphertext-Policy Attribute-Based Encryption) with hidden access control policy enables data owners to share their encrypted data using cloud storage with authorized users while keeping the access control policies blinded. However, a mechanism to prevent users from achieving successive access to a data owner's certain number of data objects, which present a conflict of interest or whose combination thereof is sensitive, has yet to be studied. In this paper, we analyze the underlying relations among these particular data objects, introduce the concept of the sensitive data set constraint, and propose a CP-ABE access control scheme with hidden attributes for the sensitive data set constraint. This scheme incorporates extensible, partially hidden constraint policy. In our scheme, due to the separation of duty principle, the duties of enforcing the access control policy and the constraint policy are divided into two independent entities to enhance security. The hidden constraint policy provides flexibility in that the data owner can partially change the sensitive data set constraint structure after the system has been set up.

1. Introduction

With the advancement of cloud computing [1], an increasing number of organizations and individual users are willing to store their private data in cloud storage to share with others. Unlike traditional access control, the data owner reserves the right to define the access control policy for his/her data for release on the cloud. Moreover, the data owner encrypts his/her data according to his/her access control policy before putting the data on the cloud because the cloud might be compromised or untrusted. Attribute-Based Encryption [2–5] provides desirable solutions to meet the data owner's needs. Compared to KP-ABE (Key-Policy Attribute-Based Encryption) [4] and fuzzy identity-based encryption [2], CP-ABE (Ciphertext-Policy Attribute-Based Encryption) [3, 5] is more appropriate, as it enables the data owner to more freely define the access control policy. Moreover, because the access control policy itself may leak critical information, efforts have been made [6–10] to hide the access control policy by blinding the attributes within it.

However, the data owner may unavoidably release some data onto the cloud storage whereby either there may exist a conflict of interest or one can derive sensitive or confidential data from the released data. For example, the data owner may release two data objects (a data object refers to an encryption data unit in CP-ABE scenario, e.g., a file, a message) $M_{\text{company\_A}}$ and $M_{\text{company\_B}}$ to the cloud, to which the Chinese Wall security policy [11] should be applied, as company_A and company_B are competitors. Any authorized user of the two data objects can freely access either data object, but once he/she has accessed one of them, he/she can no longer access the other. In the CP-ABE scenario, it is inappropriate to split all users into two mutually disjoint sets beforehand by choosing an access structure for the two data objects. This is because an authorized user is initially supposed to be able to access either of the two data objects freely. The relations among the data released to the cloud storage are substantially more complicated. Other types of data sets exist as well. In [12, 13], we emphasized the situation in which a user's consecutive access to any k data out of

n may yield a conflict of interest or disclosure of some sensitive data. This type of data set has been studied in primary access control constraints such as SOD (Separation of Duty) of RBAC (Role-Based Access Control) [14–16] and the Chinese Wall security policy [11]; we call this kind of data set a sensitive data set in this paper. The handling of the sensitive data set problem for cloud storage should be considered. Some work has been conducted on such data sets [17, 18]. However, for cloud storage, where the access control is realized via CP-ABE, there remains no such mechanism for effectively controlling a user's successive access to data objects from a data owner's sensitive data set to prevent commercial fraud, mistakes, or the leakage of critical information.

To handle the constraint for a sensitive data set stored in cloud storage that utilizes CP-ABE, we propose a CP-ABE access control scheme for sensitive data sets with a flexible, partially hidden constraint policy; in addition, the scheme retains the features of the hidden access control policy. In this work, first, we analyze the general characteristics and structures of the sensitive data set constraint and present a formal definition of it. Second, we utilize the concept of dummy attributes [19] for the data objects in the sensitive data set, and we also introduce a semitrusted constraint monitor that is responsible for keeping track of the user's access to data objects from the sensitive data set using these dummy attributes. In our scheme, the constraint policies defined by the data owner are fully hidden from entities, except for the constraint monitor; the policies are partially hidden from the constraint monitor, and the access control policies are hidden from all entities.

The user's previous successful access to data objects in a sensitive data set constraint may affect the user's future access to other data objects in the same set. To handle the sensitive data set constraint, an entity in the system needs to know whether the user has the ability to decrypt the data objects in the sensitive data set without knowing any information about the access control policy of the data object. In addition, we choose the tree access structure with "AND," "OR," and "k OF n" gates. For the above reasons, we construct our scheme based on Hur's promising scheme [8], but our emphasis is on the constraint.

The remainder of this paper is organized as follows. Section 2 explores the essential features of a sensitive data set constraint along with its structure. The assignment of dummy attributes for the sensitive data set constraint is introduced in Section 3. Section 4 provides the system architecture, assumptions, and requirements for our scheme. The formal specification of the CP-ABE access control scheme for the constraint is presented in Section 5. A detailed scheme construction is provided in Section 6. In Section 7, we discuss extra costs due to the enforcement of the constraint, in comparison with [8]. In Section 8, security, consistency, and accessibility analyses are given. In Section 9, we survey related works. Section 10 summarizes our work.

2. Sensitive Data Set Constraint

Some data objects released to the cloud storage may be characterized by certain correlations. The combination of such data objects may induce a conflict of interest; a user may easily derive highly sensitive or confidential data that are not supposed to be disclosed to any user from other legitimately accessed but otherwise insignificant data objects. Although the data owner is aware of the hazard posed by releasing such data objects to the cloud, he/she might accept the tradeoff of information protection for sharing his/her data. We first analyze the underlying relations among the data objects released by the data owner and present the formal definition of the sensitive data set constraint.

2.1. Implicit Data. $\exists M' \notin \{M_1, M_2, \ldots, M_n\}$, if all data objects from a data set $\{M_1, M_2, \ldots, M_n\}$ are accessed by one user; then, these combined access instances are equivalent to accessing data M' by this user. $\{M_1, M_2, \ldots, M_n\}$ is the minimal set that satisfies the above condition. We call M' implicit data [12].

If the implicit data are sensitive or confidential and are not supposed to be available to any user, then we should add constraints to a user's successive access to these data objects to prevent the user's derivation of the implicit data.

2.2. Structure

(a) *Chinese Wall Structure*. For the two disjoint data sets $\{M_1, M_2, \ldots, M_s\}$ and $\{M'_1, M'_2, \ldots, M'_t\}$, once a user accesses a data object from one data set, he/she should no longer be authorized to access any data object from another data set [11].

(b) (n, k) *Structure*. Denoted as $(\{M_1, M_2, \ldots, M_n\}, \#k)$, $2 \leq k \leq n$, this structure requires that no user can access k or more data objects from the data set $\{M_1, M_2, \ldots, M_n\}$. This implies that the combination of any k or more data objects from $\{M_1, M_2, \ldots, M_n\}$ is sensitive or induces a conflict of interest [13].

The Chinese Wall structure is equivalent to the (n, k) structure if we set $(n, k) = (2, 2)$ and use the concept of equivalence classes. For example, two disjoint data sets $\{M_1\}$ and $\{M_2, M_3\}$ have a Chinese Wall structure. If $[M_1] = \{M_1\}$ and $[M_2] = [M_3] = \{M_2, M_3\}$, then we have $(\{[M_1], [M_2]\}, \#2)$ or $(\{[M_1], [M_3]\}, \#2)$.

Based on the aforementioned analysis, we introduce a general and formal definition of the sensitive data set constraint.

Definition 1 (SDS constraint). Define $(\{[M_1], [M_2], \ldots, [M_n]\}, \#k)$, $2 \leq k \leq n$, as a sensitive data set (SDS) constraint if any user is forbidden from accessing data objects from k or more different equivalence classes out of n classes, and $[M_i] \cap [M_j] = \emptyset$, $0 \leq i, j \leq n$, $i \neq j$.

For an SDS constraint, we say that data objects from different equivalence classes are incompatible; on the contrary, we say that data objects from the same equivalence class are compatible. For example, for $(\{[M_1], [M_2], \ldots, [M_n]\}, \#k)$, $2 \leq k \leq n$, $\forall M'_1, M'_2$, if $M'_1 \in [M_i]$, $M'_2 \in [M_j]$, $i \neq j$, then M'_1 and M'_2 are

incompatible; if $M_1', M_2' \in [M_i]$, then M_1' and M_2' are compatible.

A data object can be under multiple SDS constraints. Assume that $(\{[M_1], [M_2], \ldots, [M_s]\}, \#k_1)$ and $(\{[M_1'], [M_2'], \ldots, [M_t']\}, \#k_2)$ are two SDS constraints; then, $(\bigcup_{i=1}^{s}[M_i]) \cap (\bigcup_{j=1}^{t}[M_j']) \neq \emptyset$ is allowed.

We also have an extra rule about SDS constraints.

Rule 1. Any two compatible data objects in an SDS constraint cannot be incompatible in any other SDS constraint and vice versa.

Let us consider an example. There is a database table of an organization; the table has columns A, B, C_1, C_2, D, and E. The data of column E for each row is sensitive or confidential; however, it can be inferred from any three corresponding columns out of A, B, C_1, D or A, B, C_2, D. The organization releases this table to the cloud storage without column E. If any user is authorized to access data of any three columns out of A, B, C_1, D or A, B, C_2, D, then he/she can infer the data of column E despite the fact that he/she is not authorized to access the data of column E [12]. So, the organization should specify an SDS constraint for this case, that is, $(\{\{M_A\}, \{M_B\}, \{M_{C_1}, M_{C_2}\}, \{M_D\}\}, \#3)$.

3. SDS Constraint Specific Attributes

In CP-ABE, attributes play a key role in the enforcement of access control. Therefore, to handle the SDS constraint, additional SDS constraint specific attributes can be used. Because the additional attributes are only used to handle the SDS constraint and have no special meaning, we adopt dummy attributes [19] in our scheme.

If a data object is organized into an SDS constraint, then we need to upgrade the original access requirement for the data object. Because the data object is no longer independent, a user's access to the data object may affect later access to other data objects under the same SDS constraint. We upgrade the original access requirements with dummy attributes. Below, we first discuss the assignment of dummy attributes for the SDS constraints.

Suppose that

$$\mathrm{SDS}_i^{(a)} = \Big(\Big\{\big\{M_{1,1}, M_{1,2}, \ldots, M_{1,n_1}\big\}, \big\{M_{2,1}, M_{2,2}, \ldots, M_{2,n_2}\big\}, \ldots, \big\{M_{l,1}, M_{l,2}, \ldots, M_{l,n_l}\big\}\Big\}, \#k\Big), \quad (1)$$

is one of the SDS constraints for data owner u_a. As shown in Figure 1, we use l dummy artificial attributes $\theta_{i,1}^{(a)}, \theta_{i,2}^{(a)}, \ldots, \theta_{i,l}^{(a)}$, which are unique to $\mathrm{SDS}_i^{(a)}$. $\theta_{i,1}^{(a)}, \theta_{i,2}^{(a)}, \ldots, \theta_{i,l}^{(a)}$ are used to distinguish between different equivalence classes. Therefore, $\theta_{i,j}^{(a)}$ corresponds to the data set $\{M_{j,1}, M_{j,2}, \ldots, M_{j,n_j}\}$ in $\mathrm{SDS}_i^{(a)}$ in this example. We assume that these attributes do not overlap with the same data owner's other dummy attributes or with the normal attributes used in the whole system.

Consider a case in which a data object is organized into two different SDS constraints. If $\mathrm{SDS}_{i_1}^{(a)} = (\{[M_1], [M_2], \ldots, [M_{l_1}]\}, \#k_1)$ and $\mathrm{SDS}_{i_2}^{(a)} = (\{[M_1'], [M_2'], \ldots, [M_{l_2}']\}, \#k_2)$, with $M \in (\bigcup_{t=1}^{l_1}[M_t]) \cap (\bigcup_{t=1}^{l_2}[M_t'])$, assume that the corresponding SDS constraint specific attributes for the two SDS constraints are $\theta_{i_1,1}^{(a)}, \theta_{i_1,2}^{(a)}, \ldots, \theta_{i_1,l_1}^{(a)}$ and $\theta_{i_2,1}^{(a)}, \theta_{i_2,2}^{(a)}, \ldots, \theta_{i_2,l_2}^{(a)}$, respectively. If $M \in [M_{j_1}]$ in $\mathrm{SDS}_{i_1}^{(a)}$ and $M \in [M_{j_2}]$ in $\mathrm{SDS}_{i_2}^{(a)}$, then $\theta_{i_1,j_1}^{(a)}$ and $\theta_{i_2,j_2}^{(a)}$ are both used to upgrade the access requirement for M.

4. CP-ABE Access Control System Architecture for SDS Constraint

4.1. System Architecture and Assumptions. The system architecture is as shown in Figure 2. The entities in the system are described as follows.

Cloud Storage Server. The cloud storage server provides the data sharing service.

Data Owner. The data owner owns the data objects and releases them to the cloud storage server after encryption under his/her access control policies. Furthermore, he/she defines the SDS constraints based on Definition 1 and Rule 1.

User (Consumer). The user accesses encrypted data objects on the cloud storage server.

Key Generation Center (KGC). The KGC generates public and private keys for the system. It is assumed to be semitrusted. The KGC performs legitimate tasks assigned to it by other entities, but it may peek at the data owner's data objects, access control policies, and constraint policies.

Proxy Server. The proxy server enforces access control for the data objects and performs partial decryption. It is assumed to be semitrusted. The proxy server performs legitimate tasks assigned to it by other entities, but it may peek at the data owner's data objects, access control policies, and constraint policies. Another additional assumption about the proxy server is that it does not tamper with any component of the ciphertext of the data object.

SDS Monitor. The SDS monitor enforces the SDS constraint for the data objects. It is assumed to be semitrusted. The SDS monitor performs legitimate tasks assigned to it by other entities, but it may peek at the data owner's data objects. Another critical assumption about the SDS monitor is that it will destroy the partially decrypted ciphertext if the user's accessing of the corresponding data object is about to violate any SDS constraint.

4.2. Security, Privacy, Consistency, and Accessibility Requirements

Security. Data confidentiality and collusion resistance [3, 8] should be guaranteed. Moreover, as per our contribution, the

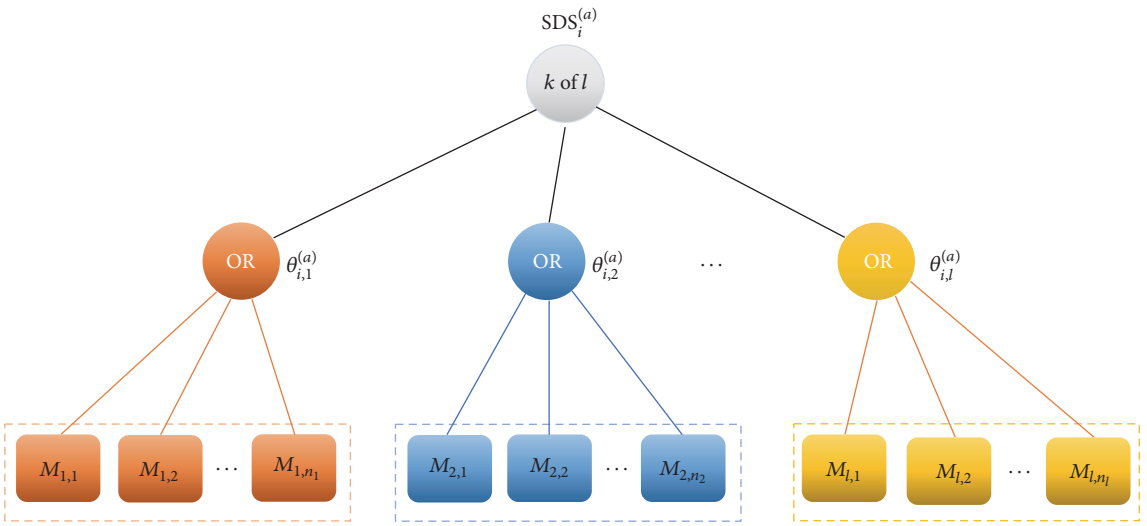

Figure 1: Dummy attributes corresponding to the data objects in an SDS constraint.

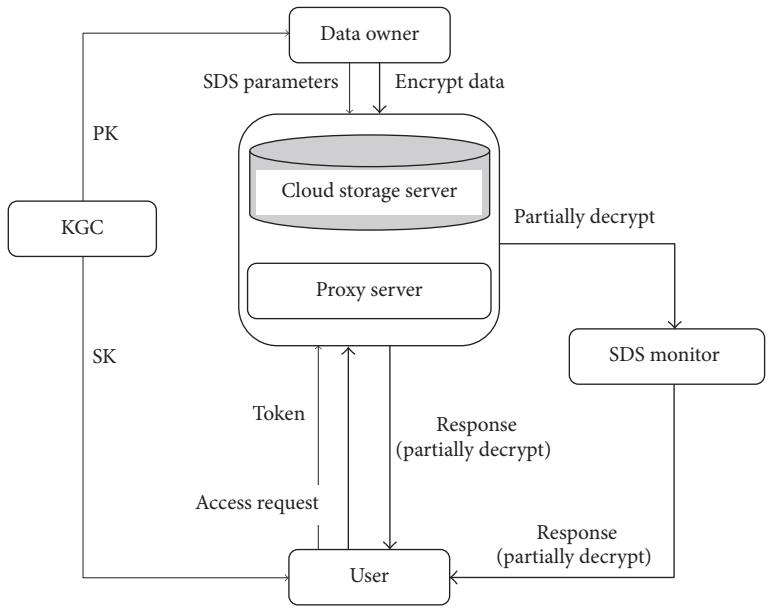

Figure 2: CP-ABE access control system architecture for SDS constraints.

SDS constraint in Definition 1 should also be guaranteed; accessing data objects should not cause a violation of any SDS constraint.

Policy Privacy. No entities have any information about the attributes of the access control policy. No entities, except for the SDS monitor, have any valuable information about the SDS constraint policy or its structure. The SDS monitor has limited information about the SDS constraint policy and its structure.

Consistency. Any two compatible data objects in an SDS constraint cannot be incompatible in any other SDS constraint and vice versa (Rule 1). The reason for the consistency requirement is simple: a user's access to data objects in an SDS constraint may affect his/her subsequent access to other data objects that are incompatible with the initial data objects being accessed but will not affect his/her subsequent access to other data objects that are compatible with the initial data objects being accessed. Thus, we have the consistency requirement to support the SDS constraint policy.

Accessibility. A user whose valid attributes match the tree access structure of a data object can access the data object even if it is organized into an SDS constraint unless the access violates any SDS constraint. In contrast, a user whose valid attributes do not match the tree access structure of the data object still cannot access the data object after it is organized into an SDS constraint.

5. CP-ABE Access Control Scheme for SDS Constraint

5.1. Cryptographic Background. In this section, we specify the formal definition of the CP-ABE access control scheme for the SDS constraint. Our proposal is based on [8]. We briefly review the cryptographic background on the reason for the integrity of the content.

5.1.1. Attribute Access Structure

Definition 2 (attribute access structure). Let $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ be a set of attributes. A collection $\mathbb{A} \subseteq 2^{\{\lambda_1, \lambda_2, \ldots, \lambda_n\}}$ is monotone if $\forall B, C$: if $B \in \mathbb{A}$ and $B \subseteq C$, then $C \in \mathbb{A}$. An attribute access structure (resp., monotone attribute access structure) is a collection (resp., monotone collection) $\mathbb{A}$ of nonempty subsets of $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$, that is, $\mathbb{A} \subseteq 2^{\{\lambda_1, \lambda_2, \ldots, \lambda_n\}} \setminus \{\emptyset\}$. We call the sets in $\mathbb{A}$ the authorized sets of attributes, and the sets not in $\mathbb{A}$ are the unauthorized sets of attributes.

The attribute access structure is a monotone access structure for all data objects regardless of whether they are in the SDS constraint.

Bilinear Pairings. $\mathbb{G}_0$ and $\mathbb{G}_1$ are two multiplicative cyclic groups of prime order p. g is a generator of $\mathbb{G}_0$, and e is a bilinear map, $e : \mathbb{G}_0 \times \mathbb{G}_0 \to \mathbb{G}_1$. e has the following properties:

(i) Bilinearity: $\forall u, v \in \mathbb{G}_o$ and $\forall x, y \in \mathbb{Z}_p^*$, $e(u^x, v^y) = e(u, v)^{xy}$.

(ii) Nondegeneracy: $e(g, g) \neq 1$.

(iii) Computability: computing the bilinear map $e : \mathbb{G}_0 \times \mathbb{G}_0 \to \mathbb{G}_1$ is efficient.

Bilinear Diffie-Hellman (BDH) Assumption. Based on the aforementioned notations, given a generator g of $\mathbb{G}_0$ and elements g^x, g^y, g^z for $x, y, z \in \mathbb{Z}_p^*$, the BDH problem is to find $e(g, g)^{xyz}$.

One-Way Anonymous Key Agreement. Kate et al. [20] proposed a one-way anonymous key agreement scheme and proved that it is secure in the random oracle model under the assumption that the BDH problem in $\langle p, \mathbb{G}_0, \mathbb{G}_1, e \rangle$ is hard with respect to unconditional anonymity, session key secrecy, and impersonation.

5.2. CP-ABE Access Control Components for the SDS Constraint. The CP-ABE access control components for the SDS constraint are as follows:

(i) USERS $= \{u_1, u_2, \ldots, u_l\}$: the user set.

(ii) O_ATTR $= \{\lambda_1, \lambda_2, \ldots, \lambda_s\}$: original attributes.

(iii) SDS_ATTR$(u_j) = \{\theta_{i,1}^{(j)}, \theta_{i,2}^{(j)}, \ldots, \theta_{i,n_i}^{(j)}\}_{1 \le i \le m}$, $j = 1, 2, \ldots, l$: SDS constraint specific attributes for a user u_j; u_j has m different SDS constraints; m can be different for different users.

(iv) SDS_INFO$_i^{(j)} = \{\theta_{i,1}^{(j)}, \theta_{i,2}^{(j)}, \ldots, \theta_{i,n_i}^{(j)}, k_i\}$, $1 \le i \le m$, $j = 1, 2, \ldots, l$: SDS constraint information for user u_j's ith SDS constraint.

(v) H_Attr$(u_t, \text{SDS}_i^{(j)})$: historical SDS constraint specific attribute set; it is initially an empty set; when user u_t successfully accesses data owner u_j's data object under u_j's ith SDS constraint, then the SDS constraint specific attribute that corresponds to the equivalence class the accessed data object belongs to will be put into the set.

(vi) SDS_ATTR $= \bigcup_{j=1}^{l}$ SDS_ATTR(u_j): all SDS constraint specific attributes in the system.

(vii) ATTR = O_ATTR $\cup$ SDS_ATTR, all attributes in the system, O_ATTR $\cap$ SDS_ATTR $= \emptyset$.

5.3. CP-ABE Access Control Scheme for SDS Constraint. The access control scheme for the SDS constraint with the hidden access control policy and the constraint policy includes the algorithms below. In our scheme, we omit the access control process for data objects that are not under the SDS constraints, as it is the same as the process in Hur's scheme [8].

Setup. Setup$(1^\lambda) \to (\text{PK}_{\text{KGC}}, \text{MK}_{\text{KGC}})$, $(\text{PK}_S, \text{MK}_S)$, $(\text{PK}_{\text{SDS}}, \text{MK}_{\text{SDS}})$. The KGC generates public parameters for the whole system. Then, the KGC, the proxy server, and the SDS monitor output their public key and master private key pairs $(\text{PK}_{\text{KGC}}, \text{MK}_{\text{KGC}})$, $(\text{PK}_S, \text{MK}_S)$, and $(\text{PK}_{\text{SDS}}, \text{MK}_{\text{SDS}})$, respectively.

Key Generation. KeyGen$(\text{MK}_{\text{KGC}}, S) \to \text{SK}_u$. The KGC takes MK_{KGC} and a set of attributes $S \in 2^{\text{O\_ATTR}} \setminus \{\emptyset\}$ as input, and it outputs the private key SK_u for the user u.

Data Encryption. Encrypt$(\text{PK}_{\text{KGC}}, \text{PK}_S, \text{PK}_{\text{SDS}}, M, \mathcal{T}) \to \text{CT}$. The data owner takes PK_{KGC}, PK_S, and PK_{SDS} and the tree access structure $\mathcal{T}$ as parameters to encrypt the data object M. The data owner outputs a ciphertext CT.

Token Generation. GenToken$(\text{SK}_u, S') \to \text{TK}_{S',u}$. The user u takes SK_u and the set of attributes $S' \subseteq S$ as input and outputs a token $\text{TK}_{S',u}$.

Partial Decryption

(a) *Proxy-Server-Side Partial Decryption.* SPartDecrypt$(\text{CT}, \text{TK}_{S',u}) \to \text{CT}'$. The proxy server determines if S' matches the access control policy. If so, then the server partially decrypts CT and outputs CT$'$ for further partial decryption by the SDS monitor; otherwise, it returns $\perp$.

(b) *SDS-Monitor-Side Partial Decryption.* SDSPartDecrypt$(\text{CT}') \to \text{CT}''$. The SDS monitor determines if the current data access violates any SDS constraint; if so, then CT$'$ is destroyed, and $\perp$ is returned; otherwise, the monitor takes CT$'$ as input and outputs CT$''$ for the user.

Data Decryption. Decrypt(CT'', SK_u) $\rightarrow M$. The user takes CT'' and SK_u as input and outputs data object M if the decryption is successful.

6. Construction of CP-ABE Access Control Scheme for SDS Constraint

The construction of the scheme is partly based on Hur's scheme [8]. However, we add the partially hidden, flexible SDS constraint policy to his scheme. To enforce the SDS constraint, the system must have the ability to determine if the user can successfully decrypt the data object before sending it to the user. This is because the user's current and previous successful access to data objects under an SDS constraint may affect the user's future access to other data objects under the same SDS constraint. In [8], the storage server happened to have the desired ability without having any knowledge about the attributes in the access structure. Therefore, Hur's work comes in handy here. The access tree specification and satisfying the access tree are the same as in [3]; we thus omit them here.

6.1. CP-ABE Access Control Scheme Construction for the SDS Constraint. Let $\mathbb{G}_0$ be a bilinear group of prime order p, and let g be the generator of $\mathbb{G}_0$. In addition, let $e : \mathbb{G}_0 \times \mathbb{G}_0 \rightarrow \mathbb{G}_1$ denote the bilinear map. A security parameter, k, will determine the size of the groups. We use two hash functions, $H : \{0,1\}^* \rightarrow \mathbb{G}_0$ and $H_1 : \mathbb{G}_1 \rightarrow \{0,1\}^{\log p}$, which we will model as a random oracle. For the Lagrange coefficients $\Delta_{i,S}$ for any $i \in \mathbb{Z}_p^*$ and a set, S, of elements in $\mathbb{Z}_p^*$, we define $\Delta_{i,S}(x) = \prod_{j \in S, j \neq i}((x-j)/(i-j))$.

Setup. The KGC, proxy server, and SDS monitor produce their public and master private key pairs.

The KGC chooses $\mathbb{G}_0$ of prime order p, where g is its generator. The KGC chooses $H : \{0,1\}^* \rightarrow \mathbb{G}_0$, $H_1 : \mathbb{G}_1 \rightarrow \{0,1\}^{\log p}$ and the following public parameters: $(\mathbb{G}_0, g, H, H_1)$.

The KGC chooses two random exponents $\alpha, \beta \in {}_R\mathbb{Z}_p^*$, $h = g^\beta$; then, the public and private keys for the KGC are $PK_{KGC} = (h, e(g,g)^\alpha)$ and $MK_{KGC} = (\beta, g^\alpha)$, respectively.

The proxy server chooses a random exponent $\gamma \in {}_R\mathbb{Z}_p^*$; then, the public and private keys for the proxy server are $PK_S = g^\gamma$ and $MK_S = H(ID_S)^\gamma$, respectively.

The SDS monitor chooses a random exponent $\sigma \in {}_R\mathbb{Z}_p^*$; then, the public and private keys for the SDS monitor are $PK_{SDS} = g^\sigma$ and $MK_{SDS} = H(ID_{SDS})^\sigma$, respectively.

Key Generation. The KGC produces private keys for users by running the KeyGen(MK_{KGC}, S) algorithm. The algorithm takes as input MK_{KGC} and S and outputs a private key for a user that holds all attributes in S. The KGC selects two random exponents $r_t \in {}_R\mathbb{Z}_p^*$ and $r_j \in {}_R\mathbb{Z}_p^*$, where r_t is a unique secret to user u_t and r_j is a unique secret to attribute $\lambda_j \in S$. Then, the KGC generates the following private key for the user:

$$SK_{u_t} = \Big(D = g^{(\alpha + r_t)/\beta},\ \forall \lambda_j \in S : D_j = g^{r_t} \cdot H(\lambda_j)^{r_j},\ D_j' = g^{r_j},\ D_j'' = H(\lambda_j)^\beta \Big). \tag{2}$$

Data Encryption. Suppose that a data owner u_a has m different SDS constraints, $SDS_i^{(a)} = (\{[M_1], [M_2], \ldots, [M_{n_i}]\}, \#k_i)$, $i = 1, 2, \ldots, m$, where the corresponding dummy attributes are $\{\theta_{i,1}^{(a)}, \theta_{i,2}^{(a)}, \ldots, \theta_{i,n_i}^{(a)}\}_{1 \leq i \leq m}$. For an SDS, the data owner selects a random $a \in {}_R\mathbb{Z}_p^*$ and then computes $K_{\theta_{i,j}} = (e(g^\sigma)^a, H(\theta_{i,j}))$, $j = 1, 2, \ldots, n_i$, $i = 1, 2, \ldots, m$. These computations can be completed beforehand. The data owner only releases the SDS constraint information $SDS\_INFO_i^{(a)} = \{\theta_{i,1}, \theta_{i,2}, \ldots, \theta_{i,n_i}, k_i\}$, $1 \leq i \leq m$, to the cloud storage server.

Suppose that the data owner u_a wants to release his/her data object $M \in \mathbb{G}_1$ to the cloud storage server, $M \in [M_j]$, where $M_j \in SDS_t^{(a)} = (\{[M_1], [M_2], \ldots, [M_{n_t}]\}, \#k_t)$. Before releasing the data object, u_a encrypts the data object under an access tree $\mathcal{T}$ defined by him/her by running the Encrypt(PK_{KGC}, PK_S, PK_{SDS}, M, $\mathcal{T}$) algorithm. The algorithm first chooses a polynomial q_x for each node x, including the leaves, in the tree $\mathcal{T}$. For additional details, see the definition of access trees in [3].

Let Y be the set of leaf nodes in $\mathcal{T}$. The data owner computes $s_y = (e(g^\beta)^a, H(\lambda_y))$ for all $y \in Y$ in the leaf node of the access tree and then computes $H_1(s_y)$.

To encrypt the data object M under $\mathcal{T}$, the data owner computes a session key between the data owner and the proxy server $K_S = e((g^\gamma)^a, H(ID_S))$ as well as the session key between the data owner and the SDS monitor $K_{SDS} = e((g^\sigma)^a, H(ID_{SDS}))$. Then, the algorithm constructs a ciphertext as

$$CT = \Big(\mathcal{T}, \widetilde{C} = M \cdot K_S \cdot K_{SDS} \cdot e(g,g)^{\alpha s}, C = h^s,\ \forall y \in Y : C_y = g^{q_y(0)},\ C_y' = H(\lambda_y)^{q_y(0)},\ H_1\left(K_{\theta_{i,j}}\right) \Big). \tag{3}$$

The data owner u_a sends (ID_a, g^a, CT) to the proxy server.

Token Generation. When a user u_t sends an access request for a data object, where the data object's owner is u_a, to the proxy server using a set of attributes $S' \subseteq S$, where S is the set of all valid attributes u_t held, user u_t obtains g^a from the proxy server first. Then, he/she generates a token TK_{S',u_t} by running the GenToken(SK_{u_t}, S') algorithm.

For all $\lambda_j \in S'$, the algorithm computes $s_j = e(g^a, D_j'') = e(g^a, H(\lambda_j)^\beta)$. Then, the algorithm selects a random $\tau \in {}_R\mathbb{Z}_p^*$ and constructs the token for S' as

$$TK_{S',u_t} = \left(\forall \lambda_j \in S' : I_j = H_1(s_j), (D_j)^\tau, (D_j')^\tau \right). \tag{4}$$

If $S' \not\subseteq S$, then the algorithm outputs $\perp$.

Next, user u_t sends the token to the proxy server and a request for the partial decryption of the ciphertext with this token. Tokens can also be precomputed.

Partial Decryption

(a) Proxy-Server-Side Partial Decryption. After receiving the token from the user, the proxy server determines whether the set of attribute indices I_j in the token matches the blinded access control policy embedded in CT. If the token matches the access control policy of the data object, then it partially decrypts the ciphertext by running the $SPartDecrypt(CT, TK_{S',u_t})$ algorithm for user u_t.

The $SPartDecrypt(CT, TK_{S',u_t})$ algorithm operates in a recursive manner. We first define a recursive algorithm $DecryptNote(CT, TK, x)$ that takes as input a ciphertext CT, a token TK, which is associated with a set of attributes S', and a node x from the tree $\mathcal{T}$. The algorithm outputs a group element of $\mathbb{G}_1$ or $\perp$.

Suppose that the proxy server runs the algorithm with a token TK_{S',u_t} provided by a user u_t. Suppose that an attribute assigned to a leaf node x is blinded as $H_1(s_x)$. Then, make the following definition: If $H_1(s_x) \in \mathcal{S}$, where $\mathcal{S}$ is a set of all attribute indices I_j associated with the token, then

$$\begin{aligned}\text{DecryptNote}\left(\text{CT}, \text{TK}_{S',u_t}, x\right) &= \frac{e\left(\left(D_x\right)^{\tau}, C_x\right)}{e\left(\left(D'_x\right)^{\tau}, C'_x\right)} \\ &= \frac{e\left(\left(g^{r_t} \cdot H\left(\lambda_x\right)^{r_x}\right)^{\tau}, g^{q_x(0)}\right)}{e\left(\left(g^{r_x}\right)^{\tau}, H\left(\lambda_x\right)^{q_x(0)}\right)} = e\left(g, g\right)^{r_t \tau q_x(0)}.\end{aligned} \tag{5}$$

If $H_1(s_x) \notin \mathcal{S}$, define $DecryptNote(CT, TK, x) = \perp$.

Now, consider the recursive case in which x is a nonleaf node. The $DecryptNote(CT, TK_{S',u_t}, x)$ algorithm is executed as follows: For all nodes z that are children of x, the algorithm calls $DecryptNote(CT, TK_{S',u_t}, z)$ and stores the output as F_z. Let S_x be an arbitrary k_x-sized set of child nodes z such that $F_z \neq \perp$; then, the algorithm computes

$$\begin{aligned}F_x &= \prod_{z \in S_x} F_z^{\Delta_{i,S'_x}(0)}, \\ &\quad \text{where } i = \text{index}\left(z\right),\ S'_x = \left\{\text{index}\left(z\right) : z \in S_x\right\} \\ &= \prod_{z \in S_x} \left(e\left(g, g\right)^{r_t \cdot \tau \cdot q_z(0)}\right)^{\Delta_{i,S'_x}(0)} \\ &= \prod_{z \in S_x} \left(e\left(g, g\right)^{r_t \cdot \tau \cdot q_{\text{parent}(z)}(\text{index}(z))}\right)^{\Delta_{i,S'_x}(0)} \\ &= \prod_{z \in S_x} \left(e\left(g, g\right)^{r_t \cdot \tau \cdot q_x(i)}\right)^{\Delta_{i,S'_x}(0)} = e\left(g, g\right)^{r_t \cdot \tau \cdot q_x(0)}\end{aligned} \tag{6}$$

and returns the result.

The algorithm begins by calling the function on the root node R of the access tree $\mathcal{T}$. If the access tree is satisfied by the token associated with the set of attributes S', then the proxy server extracts $DecryptNote(CT, TK_{S',u_t}, R) = e(g, g)^{r_t \tau s}$.

The proxy server computes $K_S = e(g^a, \text{MK}_S) = e(g^a, H(\text{ID}_S)^{\gamma})$ and $\widetilde{C}' = \widetilde{C}/K_S = M \cdot K_{\text{SDS}} \cdot e(g, g)^{\alpha s}$ in CT; then, it sends $\text{CT}' = (\widetilde{C}', C = h^s, A, H_1(K_{\theta_{t,j}}))$ to the SDS monitor, where

$$A = \text{DecryptNote}\left(\text{CT}, \text{TK}_{S',u_t}, R\right) = e\left(g, g\right)^{r_t \tau s}. \tag{7}$$

(b) SDS-Monitor-Side Partial Decryption. After receiving CT' from the proxy server, the SDS monitor determines if the current data access violates any SDS constraint defined by the data owner. If so, the SDS monitor destroys CT' and returns $\perp$, and the user's access request will be denied. Otherwise, the SDS monitor takes CT' as input and outputs CT'' for the user by running the $SDSPartDecrypt(\text{CT}')$ algorithm.

Now, we give the determination process in detail. The SDS monitor precomputes all $\Theta_{i,j} = H_1(K_{\theta_{i,j}})$, $j = 1, 2, \ldots, n_i$, $i = 1, 2, \ldots, m$, after computing $K_{\theta_{i,j}} = e(g^a, H(\theta_{i,j})^{\sigma})$, $j = 1, 2, \ldots, n_i$, $i = 1, 2, \ldots, m$, for $\{\theta_{i,1}, \theta_{i,2}, \ldots, \theta_{i,n_i}\}_{1 \le i \le m}$; and it compares $H_1(K_{\theta_{t,j}})$ in CT' to these precomputed dummy attribute indices $\Theta_{i,j}$, $j = 1, 2, \ldots, n_i$, $i = 1, 2, \ldots, m$. By comparison, it finds out that $\Theta_{t,j} = H_1(K_{\theta_{t,j}})$ and determines the SDS constraint information $\text{SDS\_INFO}_t^{(a)} = \{\theta_{t,1}^{(a)}, \theta_{t,2}^{(a)}, \ldots, \theta_{t,n_t}^{(a)}, k_t\}$, and the threshold k_t should be applied to the current data. According to the threshold value k_t, the SDS monitor checks if $|\text{H\_Attr}(u_t, \text{SDS}_t^{(a)}) \cup \{\theta_{t,j}\}| = k_t$, and then it determines whether the current data access violates SDS constraint $\text{SDS}_t^{(a)}$ and destroys the data immediately; otherwise, the SDS monitor performs its partial decryption. The SDS monitor first computes $K_{\text{SDS}} = e(g^a, \text{MK}_{\text{SDS}}) = e(g^a, H(\text{ID}_{\text{SDS}})^{\sigma})$ and then computes $\widetilde{C}'' = \widetilde{C}'/K_{\text{SDS}} = M \cdot e(g, g)^{\alpha s}$.

Next, the SDS monitor updates the historical SDS constraint specific attribute set as $\text{H\_Attr}(u_t, \text{SDS}_t^{(a)}) = \text{H\_Attr}(u_t, \text{SDS}_t^{(a)}) \cup \{\theta_{t,j}\}$ and sends $\text{CT}'' = (\widetilde{C}'', C = h^s, A)$ to the user.

Data Decryption. When user u_t receives the ciphertext CT'' from the SDS monitor, he/she uses the $\text{Decrypt}(\text{CT}'', \text{SK}_{S',u_t})$ algorithm to decrypt the ciphertext by computing

$$\begin{aligned}&\frac{\widetilde{C}''}{\left(e\left(C, D\right) / \left(A\right)^{1/\tau}\right)} \\ &= \frac{\widetilde{C}''}{\left(e\left(h^s, g^{(\alpha + r_t)/\beta}\right) / \left(e\left(g, g\right)^{r_t \tau s}\right)^{1/\tau}\right)} \\ &= \frac{\widetilde{C}''}{\left(e\left(g^{\beta s}, g^{(\alpha + r_t)/\beta}\right) / e\left(g, g\right)^{r_t s}\right)} = \frac{M e\left(g, g\right)^{\alpha s}}{e\left(g, g\right)^{\alpha s}} \\ &= M.\end{aligned} \tag{8}$$

6.2. A Case Study. We illustrate through an example how the SDS monitor checks whether the current data access violates any SDS constraint. Suppose a data owner *Alice* released three data objects M_1, M_2, and M_3 and defined tth SDS constraint

over them; that is, $\mathrm{SDS}_t^{(\mathrm{Alice})} = \{\{M_1\}, \{M_2, M_3\}, \#2\}$. The corresponding SDS constraint information $\mathrm{SDS\_INFO}_t^{(\mathrm{Alice})} = \{\theta_{t,1}, \theta_{t,2}, 2\}$ is released to the cloud storage server. $\theta_{t,1}$ and $\theta_{t,2}$ are the two dummy attributes corresponding to $\{M_1\}$ and $\{M_2, M_3\}$, respectively. Suppose a user Bob's attributes match the access structures of data objects M_1 and M_2. When Bob sends an access request for M_1, the proxy server uses Bob's token to partially decrypt the following ciphertext:

$$\begin{aligned}\mathrm{CT}_1 = \Big(&\mathcal{T}_1, \widetilde{C}_1 = M_1 \cdot K_S \cdot K_{\mathrm{SDS}} \cdot e(g,g)^{\alpha s_1},\ C_1 \\ &= h^{s_1},\ \forall y \in Y_1 : C_y = g^{q_y(0)},\ C'_y \\ &= H(\lambda_y)^{q_y(0)},\ H_1\left(K_{\theta_{t,1}}\right)\Big),\end{aligned} \tag{9}$$

where $\mathcal{T}_1$ is the tree access structure of M_1 and Y_1 is the set of leaf nodes in $\mathcal{T}_1$.

Since Bob's attributes match $\mathcal{T}_1$, then the partial decryption by the proxy server will be successful. The proxy server knows that M_1 is under an SDS constraint because there is $H_1(K_{\theta_{t,1}})$ in the ciphertext, and then it transfers the ciphertext $\mathrm{CT}'_1 = (\widetilde{C}'_1, C_1 = h^{s_1}, A_1, H_1(K_{\theta_{t,1}}))$ to the SDS monitor. When the SDS monitor receives the partially decrypted ciphertext from the proxy server, it compares $H_1(K_{\theta_{t,1}})$ in the ciphertext to the precomputed values $\Theta_{i,j} = H_1(K_{\theta_{i,j}})$, where $K_{\theta_{i,j}} = e(g^a, H(\theta_{i,j})^\sigma)$, $j = 1, 2, \ldots, n_i$, $i = 1, 2, \ldots, m$, for $\{\theta_{i,1}, \theta_{i,2}, \ldots, \theta_{i,n_i}\}_{1 \le i \le m}$, and then it gets $\Theta_{t,1} = H_1(K_{\theta_{t,1}})$. Therefore, the SDS monitor determines that $\mathrm{SDS\_INFO}_t^{(\mathrm{Alice})} = \{\theta_{t,1}, \theta_{t,2}, 2\}$ is the SDS constraint information applied to the data M_1. The SDS monitor checks if $|\mathrm{H\_Attr}(\mathrm{Bob}, \mathrm{SDS}_t^{(\mathrm{Alice})}) \cup \{\theta_{t,1}\}| = 2$. Since $\mathrm{H\_Attr}(\mathrm{Bob}, \mathrm{SDS}_t^{(\mathrm{Alice})}) = \emptyset$, we have

$$\left|\mathrm{H\_Attr}\left(\mathrm{Bob}, \mathrm{SDS}_t^{(\mathrm{Alice})}\right) \cup \{\theta_{t,1}\}\right| = 1 < 2. \tag{10}$$

Therefore, the SDS monitor knows that the current access to M_1 does not violate $\mathrm{SDS}_t^{(\mathrm{Alice})}$. After that, it updates $\mathrm{H\_Attr}(\mathrm{Bob}, \mathrm{SDS}_t^{(\mathrm{Alice})})$ as

$$\begin{aligned}&\mathrm{H\_Attr}\left(\mathrm{Bob}, \mathrm{SDS}_t^{(\mathrm{Alice})}\right) \\ &\quad = \mathrm{H\_Attr}\left(\mathrm{Bob}, \mathrm{SDS}_t^{(\mathrm{Alice})}\right) \cup \{\theta_{t,1}\}\end{aligned} \tag{11}$$

and then sends $\mathrm{CT}''_1 = (\widetilde{C}''_1, C_1 = h^{s_1}, A_1)$ to Bob. Bob successfully accesses M_1.

When Bob sends another access request for M_2, the proxy server uses Bob's token to partially decrypt the following ciphertext:

$$\begin{aligned}\mathrm{CT}_2 = \Big(&\mathcal{T}_2, \widetilde{C}_2 = M_2 \cdot K_S \cdot K_{\mathrm{SDS}} \cdot e(g,g)^{\alpha s_2},\ C_2 \\ &= h^{s_2},\ \forall y \in Y_2 : C_y = g^{q_y(0)},\ C'_y \\ &= H(\lambda_y)^{q_y(0)},\ H_1\left(K_{\theta_{t,2}}\right)\Big),\end{aligned} \tag{12}$$

where $\mathcal{T}_2$ is the tree access structure of M_2 and Y_2 is the set of leaf nodes in $\mathcal{T}_2$.

Since Bob's attributes match $\mathcal{T}_2$, then the partial decryption by the proxy server will be successful. The proxy server knows that the ciphertext is under an SDS constraint because there is $H_1(K_{\theta_{t,2}})$, and then it transfers the ciphertext $\mathrm{CT}'_2 = (\widetilde{C}'_2, C_2 = h^{s_2}, A_2, H_1(K_{\theta_{t,2}}))$ to the SDS monitor. When the SDS monitor receives the partially decrypted ciphertext from the proxy server, it compares $H_1(K_{\theta_{t,2}})$ in the ciphertext to the precomputed values $\Theta_{i,j} = H_1(K_{\theta_{i,j}})$, where $K_{\theta_{i,j}} = e(g^a, H(\theta_{i,j})^\sigma)$, $j = 1, 2, \ldots, n_i$, $i = 1, 2, \ldots, m$, for $\{\theta_{i,1}, \theta_{i,2}, \ldots, \theta_{i,n_i}\}_{1 \le i \le m}$, and then it gets $\Theta_{t,2} = H_1(K_{\theta_{t,2}})$. Therefore, the SDS monitor determines that $\mathrm{SDS\_INFO}_t^{(\mathrm{Alice})} = \{\theta_{t,1}, \theta_{t,2}, 2\}$ is also the SDS constraint information applied to the data M_2. The SDS monitor checks if $|\mathrm{H\_Attr}(\mathrm{Bob}, \mathrm{SDS}_t^{(\mathrm{Alice})}) \cup \{\theta_{t,2}\}| = 2$. Now, we have

$$\left|\mathrm{H\_Attr}\left(\mathrm{Bob}, \mathrm{SDS}_t^{(\mathrm{Alice})}\right) \cup \{\theta_{t,2}\}\right| = 2, \tag{13}$$

so the SDS monitor knows that the current access to M_2 violates $\mathrm{SDS}_t^{(\mathrm{Alice})}$. It destroys the corresponding ciphertext immediately. Bob cannot access M_2.

7. Extra Costs due to the SDS Constraint and Comparison

We discuss the additional costs due to the enforcement of the SDS constraint, in comparison with [8]. We also compare our scheme with [9, 10]. We use notations in the Notations Section.

In Table 1, we compare the user's private key size, ciphertext size, public key size, and computational cost used in the whole system with those in [8–10]. Compared to [8], the user's private key size is the same, the ciphertext size is increased by the bit size of an element in $\mathbb{G}_1$ because $H_1(K_{\theta_{i,j}})$ is embedded into it, and the public key size is increased by the bit size of an element in $\mathbb{G}_0$ because $\mathrm{PK}_{\mathrm{SDS}} = g^\sigma$ is used as one of the public keys in the system.

In addition, we have the additional dummy attributes $\{\theta_{i,1}, \theta_{i,2}, \ldots, \theta_{i,n_i}\}_{1 \le i \le m}$ for a data owner who has m different SDS constraints. However, the obfuscation of these dummy attributes by the data owner and the computation of the attribute indices of these dummy attributes by the SDS monitor can be performed beforehand.

Compared to [8–10], our scheme has an extra network communication cost per data object if the data object is under an SDS constraint. The proxy server sends the partially decrypted ciphertext to the SDS monitor, and the SDS monitor sends the further partially decrypted ciphertext to the user.

8. Security, Consistency, and Accessibility Analysis

The security, policy privacy, consistency, and accessibility analyses are presented in this section with respect to the requirements enumerated in Section 4.2.

Table 1: Comparison of different schemes.

	Hur's [8]	Lai et al.'s [9]	Lai et al.'s [10]	The proposed scheme
Ciphertext size	$(2t+1)L_0 + L_1$	$(N+1)L_0 + L_1$	$(4\ell+2)L_0 + 2L_1$	$(2t+1)L_0 + 2L_1$
Private key size	$(3k+1)L_0$	$(u+1)L_0$	$(u+2)L_0$	$(3k+1)L_0$
Public key size	$2L_0 + L_1$	$(N+2)L_0 + L_1$	$(u+5)L_0 + L_1$	$3L_0 + L_1$
Encryption cost	$(3t+3)\text{exp} + (t+1)\widehat{e}$	$(N+2)\text{exp}$	$(8\ell+4)\text{exp}$	$(3t+4)\text{exp} + (t+2)\widehat{e}$
Partial decryption cost (storage server or proxy server)	$(2\lvert R\rvert+1)\widehat{e} + \text{exp}$	/	/	$(2\lvert R\rvert+1)\widehat{e} + \text{exp}$
Partial decryption cost (SDS monitor)	/	/	/	$\widehat{e} + \text{exp}$
Decryption cost (user)	$\widehat{e} + 3\,\text{exp}$	$(N+1)\widehat{e}$	$(4\ell+2)\widehat{e} + (2\ell+3)\text{exp}$	$\widehat{e} + 3\,\text{exp}$
Token generation cost (user)	$\lvert R\rvert\widehat{e} + 2\lvert R\rvert\text{exp}$	/	/	$\lvert R\rvert\widehat{e} + 2\lvert R\rvert\text{exp}$

8.1. Security

Data Confidentiality. For data objects that are not under any SDS constraint, confidentiality is guaranteed because we follow [8] for these data objects. The SDS monitor can just be regarded as an external user who does not have sufficient attributes.

Considering data objects that are under the SDS constraints, we introduced the following concepts: the SDS monitor, the SDS constraint specific dummy attributes, and additional partial decryption by the SDS monitor. Therefore, we mainly analyze the effect of the corresponding changes regarding the data confidentiality.

An external user u_t whose attributes do not match the tree access structure in the ciphertext cannot produce a valid token for partial decryption. This results in the fact that the proxy server cannot extract the expected value $e(g,g)^{r_t \tau s}$ with the invalid token without knowing r_t and τ, which are unique secrets to the user.

Assuming that the unauthorized user u_t directly fetches the ciphertext from the cloud storage server without using the token, he/she cannot compute $e(g,g)^{r_t s}$ because his/her attributes do not match the access tree. Moreover, he/she cannot cast off the session keys K_S and K_{SDS} embedded in the ciphertext component $\widetilde{C} = M \cdot K_S \cdot K_{\text{SDS}} \cdot e(g,g)^{\alpha s}$ because K_S is only shared by the data owner and the proxy server, and K_{SDS} is only shared by the data owner and the SDS monitor.

The proxy server, similar to other external unauthorized users, not only does not have sufficient attributes to decrypt the ciphertext but also cannot cast off K_{SDS} embedded in the ciphertext component $\widetilde{C} = M \cdot K_S \cdot K_{\text{SDS}} \cdot e(g,g)^{\alpha s}$ because K_{SDS} is only shared by the data owner and the SDS monitor.

The KGC cannot decrypt the ciphertext because the session keys K_S and K_{SDS} are embedded in the ciphertext component $\widetilde{C} = M \cdot K_S \cdot K_{\text{SDS}} \cdot e(g,g)^{\alpha s}$. In addition, K_S is shared only by the data owner and the proxy server, and K_{SDS} is shared only by the data owner and the SDS monitor. The KGC cannot determine K_S and K_{SDS} because of the session key secrecy property of the key agreement [20]. Assuming that the KGC can access the partially decrypted ciphertext $\text{CT}' = (\widetilde{C}', C = h^s, A, H_1(K_{\theta_{i,j}}))$, where $\widetilde{C}' = \widetilde{C}/K_S = M \cdot K_{\text{SDS}} \cdot e(g,g)^{\alpha s}$ from the proxy server, it still cannot cast off K_{SDS} because K_{SDS} is only shared by the data owner and the SDS monitor. Moreover, the KGC cannot cast off τ from $A = e(g,g)^{r_t \tau s}$ to obtain the expected value $e(g,g)^{r_t s}$ because τ is a unique secret to the user.

The SDS monitor cannot decrypt the ciphertext for two reasons. First, it cannot cast off τ from $A = e(g,g)^{r_t \tau s}$ to obtain the expected value $e(g,g)^{r_t s}$ because τ is a unique secret to the user. Second, the SDS monitor does not know β, r_t, or g^α to obtain the expected value $g^{(\alpha + r_t)/\beta}$ because β and g^α are private keys of the KGC and because r_t is a unique secret to the user.

In addition, the SDS constraint specific dummy attributes themselves do not disclose any information about the content of the data object because they are completely independent of the content of the data object.

Collusion Resistance. For both ordinary data objects and the data objects that are under the SDS constraints, collusion resistance is guaranteed. The ciphertext component $\widetilde{C} = M \cdot K_S \cdot K_{\text{SDS}} \cdot e(g,g)^{\alpha s}$ is different from that in Hur's scheme for the data that are under an SDS constraint, but this does not affect collusion resistance. The random value r_t, which is unique to each user in the users' private keys, prevents several users from combining their private keys to produce a token to decrypt the ciphertext unless one of the users has sufficient valid attributes to produce a token to achieve $e(g,g)^{\alpha s}$.

SDS Constraint. The data object that is under an SDS constraint must pass through the SDS constraint checkpoint, the SDS monitor, because the session key K_{SDS} is embedded in the ciphertext component $\widetilde{C} = M \cdot K_S \cdot K_{\text{SDS}} \cdot e(g,g)^{\alpha s}$. Neither the proxy server, the KGC, nor the user can cast off K_{SDS} because K_{SDS} is only shared by the data owner and the SDS monitor. When the SDS monitor receives the partially decrypted ciphertext from the proxy server, the SDS monitor checks if the user's current data access violates an SDS constraint by comparing $H_1(K_{\theta_{i,j}})$ in the ciphertext to the precomputed values $\Theta_{i,j} = H_1(K_{\theta_{i,j}})$, $j = 1, 2, \ldots, n_i$, $i = 1, 2, \ldots, m$, where $K_{\theta_{i,j}} = e(g^a, H(\theta_{i,j})^\sigma)$, $j = 1, 2, \ldots, n_i$, $i = 1, 2, \ldots, m$, for $\text{SDS\_INFO}_i^{(a)} = \{\theta_{i,1}^{(a)}, \theta_{i,2}^{(a)}, \ldots, \theta_{i,n_i}^{(a)}, k_i\}_{1 \le i \le m}$, and it checks if $\lvert \text{H\_Attr}(u_t, \text{SDS}_t^{(u_a)}) \cup \{\theta_{t,j}\}\rvert = k_t$. If the SDS monitor determines that the current data access violates an SDS constraint, the SDS monitor destroys the data immediately. We consider that the duty of access control policy enforcement and the duty of SDS constraint policy

enforcement should be separated into two different entities. Therefore, we add an SDS monitor to the system architecture instead of only using the proxy server to perform both duties. Performing this separation follows the access control principle of separation of duty.

8.2. Policy Privacy

Access Control Policy Privacy. For both ordinary data object and the data objects that are under the SDS constraint, access control policy privacy is guaranteed because we follow Hur's scheme [8]. We omit the description here.

Constraint Policy Privacy. First, when the data owner only releases the SDS constraint information $\{\theta_{i,1}, \theta_{i,2}, \ldots, \theta_{i,n_i}, k_i\}_{1 \le i \le m}$ to the cloud storage server, he/she does not disclose any information about which dummy attributes are used for which data object. Initially, other entities only have knowledge that $\{\theta_{i,1}, \theta_{i,2}, \ldots, \theta_{i,n_i}, k_i\}_{1 \le i \le m}$ are used for SDS constraint and nothing else.

If the user receives the partially decrypted ciphertext from the SDS monitor, and not from the proxy server, then he/she only knows that the current data object is under an SDS constraint. The user does not know to which SDS constraint the data object belongs; the relations among this data object and other data objects received from the SDS monitor previously are also unknown to him/her.

The KGC and the proxy server know that the ciphertext is under an SDS constraint if there is a ciphertext component $H_1(K_{\theta_{i,j}})$. If they observe that several ciphertexts that belong to a data owner have the same $H_1(K_{\theta_{i,j}})$, then they only know that the corresponding data objects belong to the same equivalence class in an SDS; this does not pose a security threat. However, they still do not know other structure information of an SDS constraint, including the threshold k_i, and other data objects that are in a different equivalence class of the same SDS constraint.

The SDS monitor can observe the relations among the ciphertexts but does not know the entire SDS structure exactly. For example, it does not know how many data objects are contained in the same equivalence class. The total number of equivalence classes can vary over time as the number of corresponding dummy attributes varies. However, the SDS monitor is responsible for enforcing the SDS constraint; therefore, the SDS monitor is the entity that possesses the most knowledge about the SDS constraint structure; this situation is indispensable.

8.3. Consistency. Any two compatible data objects under an SDS constraint cannot be incompatible under any other SDS constraint and vice versa. The data owner is required to define consistent SDS constraints. This can be achieved as follows. For each data object M, we maintain its global current incompatible set $\overline{M} = \{M' \mid \exists \text{SDS}, M, M' \in \text{SDS} \wedge M' \notin [M]\}$. If the data owner needs to add a new SDS constraint or add new data objects into an existing SDS constraint that includes the data object M, then he/she should ensure that data objects from $\overline{M}$ are not put into $[M]$. The details on how to maintain SDS constraint consistency are beyond the scope of this paper. However, if we cannot guarantee this consistency, this may result in a violation of both security and accessibility requirements.

8.4. Accessibility. A user whose valid attributes match the tree access structure of a data object can access the data object even if it is organized into an SDS constraint, unless the data access violates an SDS constraint. Access control policy enforcement is implemented by the proxy server using the user's token, which is associated with the user's valid attributes; thus, the proxy server can partially decrypt the ciphertext if the user has the valid attributes. The partially decrypted ciphertext must pass through the SDS monitor to enforce the SDS constraint. If the current data access does not violate the SDS constraint, then the SDS monitor also partially decrypts the ciphertext to send it to the user for full decryption; otherwise, the SDS monitor destroys it, and the user cannot receive the partially decrypted ciphertext. Therefore, this requirement is guaranteed to be met.

In contrast, a user whose valid attributes do not match the tree access structure of a data object still cannot access the data object after it is organized into an SDS constraint. Access control policy enforcement is performed first by the proxy server using the user's token, which is associated with the user's valid attributes. Therefore, the proxy server cannot partially decrypt the ciphertext if the user does not possess valid attributes. In that case, the proxy server does not pass the ciphertext to either the SDS monitor or the user; the user thus cannot access the data object. Therefore, this requirement is guaranteed to be met.

9. Related Work

There are quite a few research works on the formal specification of access control constraints [13, 16, 17, 21–24]. Crampton [16] analyzed and classified RBAC constraints in detail. In our previous works [13, 24], we proposed general access control constraints against similar users' successive access to permissions from a sensitive combination of permissions. Joshi et al. [21] presented the formal specification of RBAC constraints, therein considering time and location in access control decision-making. Sharifi and Tripunitara [17] proposed an approach for enforcing the Chinese Wall security policy in a least-restrictive manner compared to Brewer and Nash's specification [11]. Bijon et al. [23] first introduced ABCL (Attribute-Based Constraint Specification Language) to specify constraints in ABAC (Attribute-Based Access Control), which supports the SOD of RBAC. In contrast to these works, we intended to generalize access control constraints for data released to a cloud storage server, on which the promising CP-ABE access control mechanism is utilized; we handled the constraint in a less-restrictive manner based on accessibility requirements. Our proposal coincides as much as possible with the CP-ABE access control paradigm. The SDS constraint proposed in this paper is a compact and generalized constraint that mostly covers the SOD constraint and Chinese Wall security policy. However, we did not consider *write* actions on the data by the user

(consumer); the main action described in this paper is the *read* action.

In data outsourcing environments, access control is supported by cryptographic methods [25–29]. Di Vimercati et al. [25, 26] introduced a novel approach that combines cryptography with access control. Data are placed with an honest but curious third party; therefore, their approach adopted a two-layer encryption method; the first layer of encryption is performed by the data owner, as the server is not fully trusted, and the server performs the second layer of encryption to reflect dynamic changes over the access control policy. Asghar et al. used RBAC policies for outsourced environments [28]. In their work, the actual RBAC policy is hidden via encryption from the service provider, as the service provider is honest but curious. The PDP (Policy Decision Point) can perform the policy evaluation without disclosing the contents of the requests or policies to the service provider. However, [25–28] did not consider access control constraints. Compared to [25–28], our emphasis is placed on the generalization and enforcement of access control constraints, which covers most of static SOD and Chinese Wall security policies, on outsourced data released to the cloud with the objective of achieving better synergy with the CP-ABE access control mechanism.

The privacy of the access control policy defined for the outsourced data, where CP-ABE realizes the access control, is also a nontrivial issue. Researchers have been focusing on hiding access control policies from outsiders [6–10]. Yu et al. [6] and Nishide et al. [7] proposed CP-ABE schemes with a hidden access control policy. Their schemes only used "AND" gates, which restrains the expressiveness of the policy; the key size of each user is proportional to the number of attributes. In Nishide et al.'s second scheme [7], new possible values for attributes can be added after system setup. Hur's work [8] is also a variant of CP-ABE with a hidden access control policy. In his work, the computation-intensive work, the partial decryption of the ciphertext, is completed by the storage center; the access control policy has the same expressiveness as in Bethencourt et al.'s work [3]. The main reason why we extend Hur's scheme is because the handling of the SDS constraint in our approach needs an entity, aside from the data owner, to determine if the user's attributes satisfy the access control policy of the data object before sending the data object to the user without knowing the policy; in Hur's scheme, the storage center possesses the desired ability. Lai et al. [9, 10] proposed CP-ABE with a partially hidden access policy. In their work, the attributes have two parts: the attribute name and the attribute value. The data owner hides the attribute value, and thus the access policy is partially hidden. Their schemes are fully secure in the standard model. Compared to these works, we followed CP-ABE with the hidden access control policy; however, our emphasis focused on how to handle the SDS constraint, where the SDS constraint policy structure is hidden from all entities except for the SDS monitor. The constraint policy structure is partially hidden from the SDS monitor.

Asghar et al. [29] proposed a cryptographic approach for enforcing the dynamic SOD and Chinese Wall security policy. Their proposal hides users' access histories with respect to dynamic SOD via encryption because a user's access history might reveal critical information to the service provider, who is not fully trusted. They utilized Bethencourt et al.'s [3] access structure to describe the access control constraint. Their work is on hiding conventional constraints. Compared to their work, our work is suitable for outsourced data access control realized by CP-ABE with a hidden access control policy. In our work, the constraint policy related attributes are blinded and embedded in the ciphertext; most of the computations related to the constraint policy can be precomputed, and the SDS constraint policy can be partially changed after system setup. We can easily extend our scheme by adding a time duration to transform the SDS constraint into a time-aware dynamic constraint.

10. Conclusion

In this paper, we proposed an access control approach for generalized SDS constraints for cloud storage; the constraint originated from the SOD of RBAC and the Chinese Wall security policy. Our approach is based on CP-ABE with a hidden access control policy. We handled the SDS constraint using additional artificial attributes through the participation of an additional entity: the SDS monitor. To enforce the SDS constraint, a session key, established between the data owner and the SDS monitor, is embedded in the ciphertext component to force the proxy server to pass the initial partially decrypted ciphertext to the SDS monitor to perform the second partial decryption. The SDS constraint policy structure is hidden from the KGC, the proxy server, and the user. To prevent commercial fraud or mistakes, the duties of enforcing both the access control policy and the constraint policy are divided between the proxy server and the SDS monitor. The security, policy privacy, consistency, and accessibility analyses indicated that the approach is secure and effective. The SDS constraint policy is also extensible in that the data owner puts more data objects into the equivalence classes of an SDS constraint after system setup; the data owner can increase the total number of equivalence classes and the threshold in an SDS constraint at a lower cost following system setup. To our knowledge, the generalized concept of SDS constraints and their structure as well as the use of dummy attributes to enforce a partially hidden constraint policy represents the novelty of our work.

As our future work, we will consider how to fully hide the constraint policy structure from all entities, especially from the SDS monitor, without affecting the enforcement of the SDS constraint. To improve the expressiveness of the SDS constraint, we will also consider using "AND," "OR," and "k OF n" in the SDS constraint.

Notations

L_0: Bit-length of element in $\mathbb{G}_0$
L_1: Bit-length of element in $\mathbb{G}_1$
t: The number of attributes associated with the ciphertext
k: The number of attributes associated with the private key of a user
u: The size of original attribute universe

N: The total number of possible values of all attributes
$|R|$: The number of user's attributes satisfying an access structure
ℓ: The number of rows in LSSS matrix
exp: Exponentiation in group operation
$\widehat{e}$: Bilinear pairing.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors would like to thank Professor David Du at the Department of Computer Science and Engineering, University of Minnesota, for providing the necessary support to conduct the research and also for providing valuable suggestions. This research work was supported by the National Natural Science Foundation of China (Grants nos. 61562085, 11261057, and 11461069).

References

[1] P. Mell et al., "The NIST definition of cloud computing," 2011.

[2] A. Sahai and B. Waters, "Fuzzy Identity-Based Encryption," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 457–473, Springer, Berlin, Germany, 2005.

[3] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proceedings of the IEEE Symposium on Security and Privacy (SP '07)*, pp. 321–334, May 2007.

[4] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS '06)*, pp. 89–98, November 2006.

[5] B. Waters, "Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization," in *Proceedings of the International Workshop on Public Key Cryptography*, pp. 53–70, Springer, Berlin, Germany, 2011.

[6] S. Yu, K. Ren, and W. Lou, "Attribute-based content distribution with hidden policy," in *Proceedings of the 4th IEEE Workshop on Secure Network Protocols, NPSec'08*, pp. 39–44, October 2008.

[7] T. Nishide, K. Yoneyama, and K. Ohta, "Attribute-based encryption with partially hidden encryptor-specified access structures," in *Proceedings of the International Conference on Applied Cryptography and Network Security*, pp. 111–129, Springer, Berlin, Germany, 2008.

[8] J. Hur, "Attribute-based secure data sharing with hidden policies in smart grid," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 11, pp. 2171–2180, 2013.

[9] J. Lai, R. H. Deng, and Y. Li, "Fully secure cipertext-policy hiding CP-ABE," in *Proceedings of the International Conference on Information Security Practice and Experience*, pp. 24–39, Springer, Berlin, Germany, 2011.

[10] J. Lai, R. H. Deng, and Y. Li, "Expressive CP-ABE with partially hidden access structures," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security (ASIACCS '12)*, pp. 18-19, ACM, Seoul, Republic of Korea, May 2012.

[11] D. F. C. Brewer and M. J. Nash, "The Chinese Wall security policy," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 206–214, IEEE Computer Society Press, Los Alamitos, CA, USA, May 1989.

[12] Q. Tayir, K. Rahman, and N. Helil, "Risky permission set based access control constraint," in *Proceedings of the 2015 International Conference on Computer Science And Technology (ICCST '15)*, 517, 510 pages, 2015.

[13] N. Helil and K. Rahman, "Secret sharing scheme based approach for access control constraint against similar users' collusive attack," *JISE. Journal of Information Science and Engineering*, vol. 32, no. 6, pp. 1455–1470, 2016.

[14] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[15] D. F. Ferraiolo, R. S. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Transactions on Information and System Security*, vol. 4, no. 3, pp. 224–274, 2001.

[16] J. Crampton, "Specifying and enforcing constraints in role-based access control," in *Proceedings of Eighth ACM Symposium on Access Control Models and Technologies*, pp. 43–50, June 2003.

[17] A. Sharifi and M. V. Tripunitara, "Least-restrictive enforcement of the chinese wall security policy," in *Proceedings of the 18th ACM Symposium on Access Control Models and Technologies, SACMAT '13*, pp. 61–72, June 2013.

[18] R. Wu, G.-J. Ahn, H. Hu, and M. Singhal, "Information flow control in cloud computing," in *Proceedings of the 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom '10*, pp. 1–7, IEEE, Ottawa, ON, Canada, October 2010.

[19] M. Chase, "Multi-authority Attribute Based Encryption," in *Proceedings of the Theory of Cryptography Conference*, pp. 515–534, 2007.

[20] A. Kate, G. Zaverucha, and I. Goldberg, "Pairing-Based Onion Routing," in *Proceedings of the International Workshop on Privacy Enhancing Technologies*, pp. 95–112, Springer, Berlin, Germany, 2007.

[21] J. B. D. Joshi, E. Bertino, U. Latif, and A. Ghafoor, "A generalized temporal role-based access control model," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 1, pp. 4–23, 2005.

[22] D. Basin, S. J. Burri, and G. Karjoth, "Separation of duties as a service," in *Proceedings of the 6th International Symposium on Information, Computer and Communications Security, ASIACCS 2011*, pp. 423–429, March 2011.

[23] K. Z. Bijon, R. Krishman, and R. Sandhu, "Constraints specification in attribute based access control," *Science*, vol. 2, pp. 131–144, 2013.

[24] N. Helil and K. Rahman, "Attribute based access control constraint based on subject similarity," in *Proceedings of the 2014 IEEE Workshop on Advanced Research and Technology in Industry Applications, WARTIA '14*, pp. 226–229, IEEE, Ottawa, ON, Canada, September 2014.

[25] S. D. C. Di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Over-encryption: management of access control evolution on outsourced data," in *Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB '07*, pp. 123–134, VLDB endowment, Washington, Wash, USA, September 2007.

[26] S. D. C. Di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Encryption policies for regulating access to outsourced data," *ACM Transactions on Database Systems*, vol. 35, pp. 12:1–12:46, 2010.

[27] M. R. Asghar, M. Ion, G. Russello, and B. Crispo, "ESPOON: Enforcing encrypted security policies in outsourced environments," in *Proceedings of the 2011 6th International Conference on Availability, Reliability and Security, ARES '11*, pp. 99–108, IEEE, Vienna, Austria, August 2011.

[28] M. R. Asghar, M. Ion, G. Russello, and B. Crispo, "ESPOON ERBAC: enforcing security policies in outsourced environments," *Computers and Security*, vol. 35, pp. 2–24, 2013.

[29] M. R. Asghar, G. Russello, and B. Crispo, "E-GRANT: enforcing encrypted dynamic security constraints in the cloud," in *Proceedings of the 3rd International Conference on Future Internet of Things and Cloud, FiCloud '15*, pp. 135–144, IEEE, Rome, Italy, August 2015.